浙江大学

本科实验报告

课程名称：计算机组成
姓名：曾一欣

学院：竺可桢学院

班级：求是科学班（计算机科学与技术）

专业：计算机科学与技术

学号：3180105144

指导老师：姜晓红
2020.6.19

浙江大学实验报告

课程名称： 计算机组成　　实验类型：综合

实验项目名称：Lab4 Multi Cycle CPU

学生姓名：曾一欣　　专业：计算机科学与技术　　学号： 3180105144

同组学生姓名： None　　指导老师： 姜晓红

实验地点：无　　实验日期： 2020 年 6 月 19 日

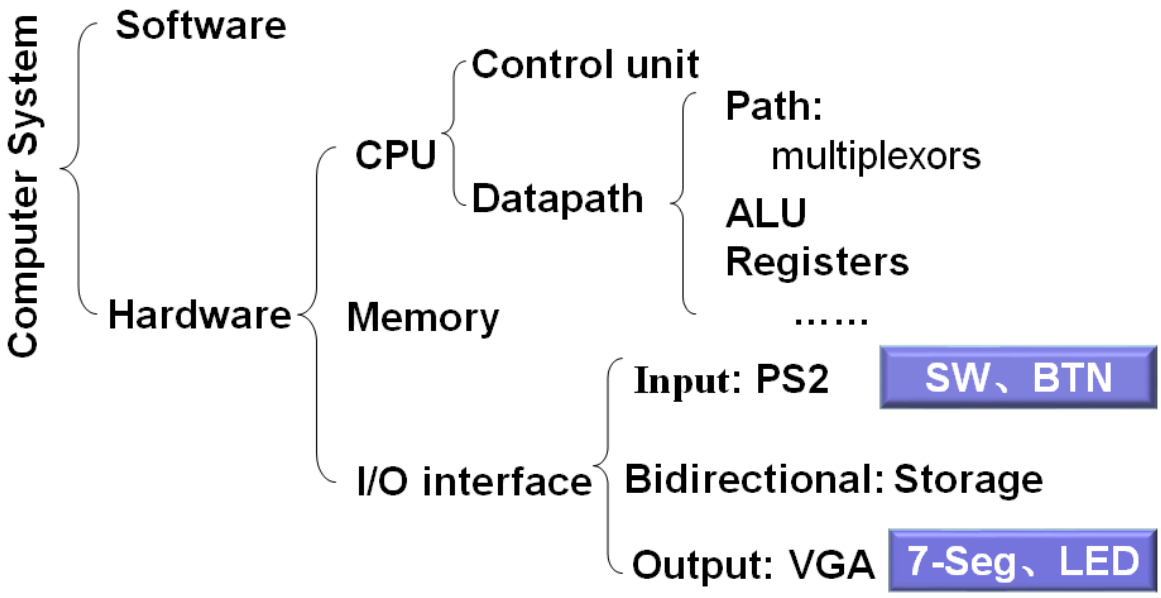
note: In this experiment I designed the SCPU by myself instead of follow the instructions in the slides. So the report might be different from lab PPTs because my steps to construct the SCPU is different.

一、 实验目的和要求

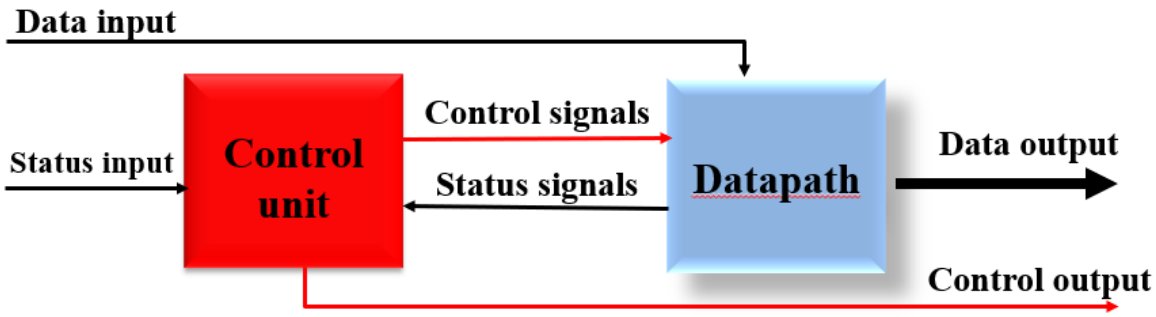
1. 构建数据通路
2. 构建控制器
3. 设计数据通路的核心部件
4. 连接完成MCPU

二、 实验内容和原理

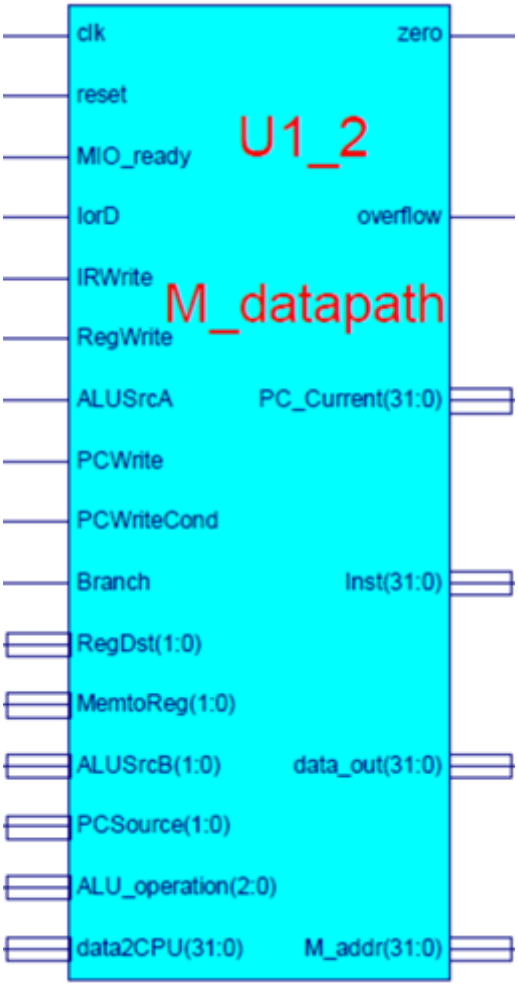
Decomposability of computer systems



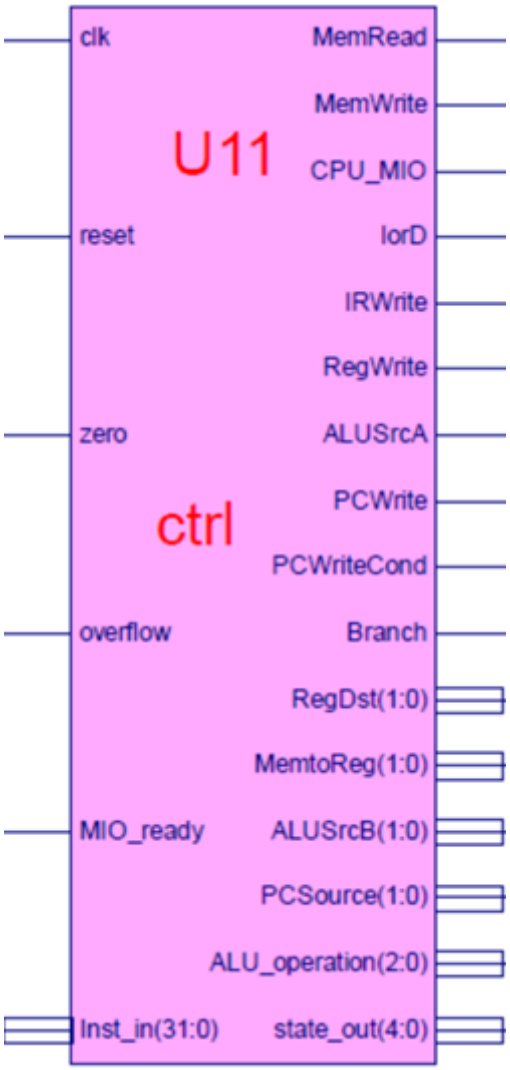
Digital circuit



DataPath



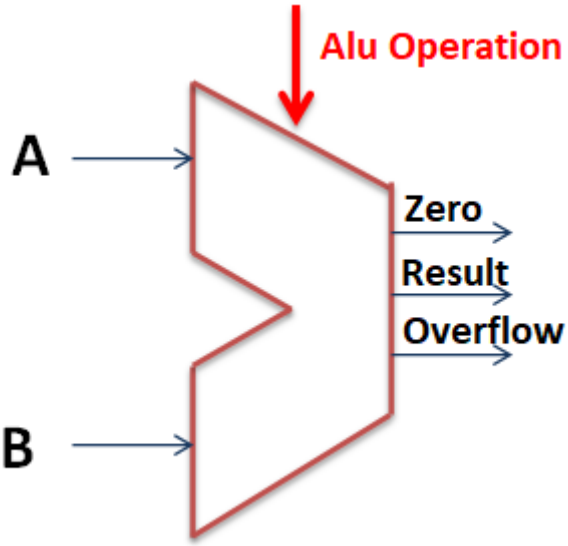
Controller



ALU

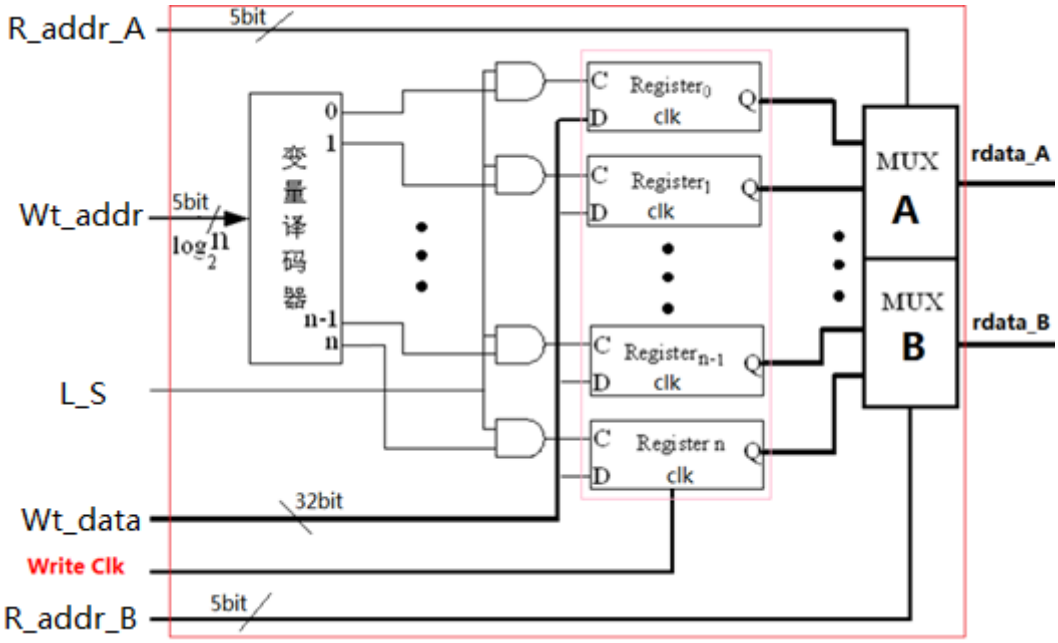
use the same module ALU in SCPU

ALU Control Lines	Function	note
000	And	兼容
001	Or	兼容
010	Add	兼容
110	Sub	兼容
111	Set on less than	
100	nor	扩展
101	srl	扩展
011	xor	扩展



Register file

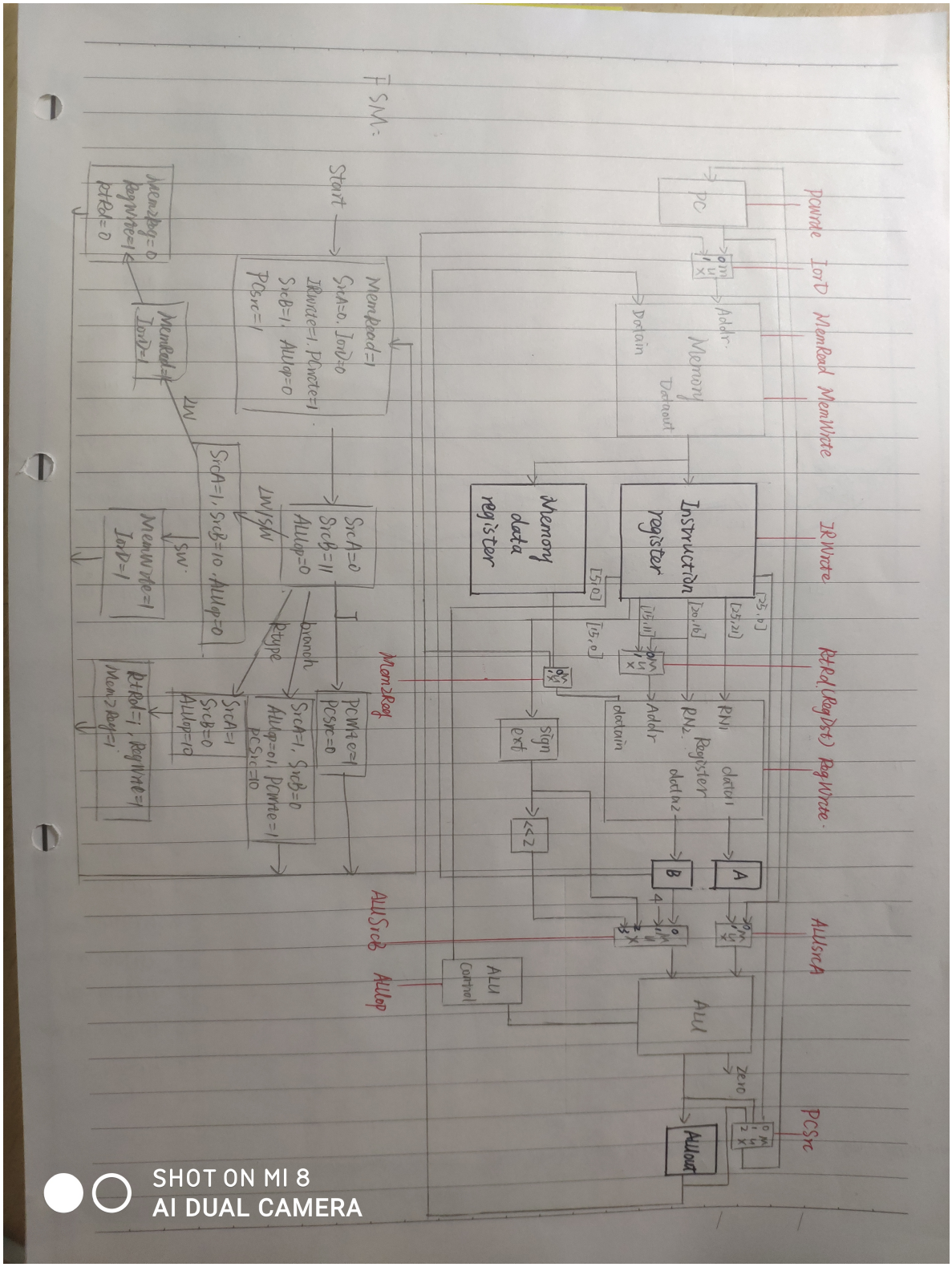
use the same module in SCPU



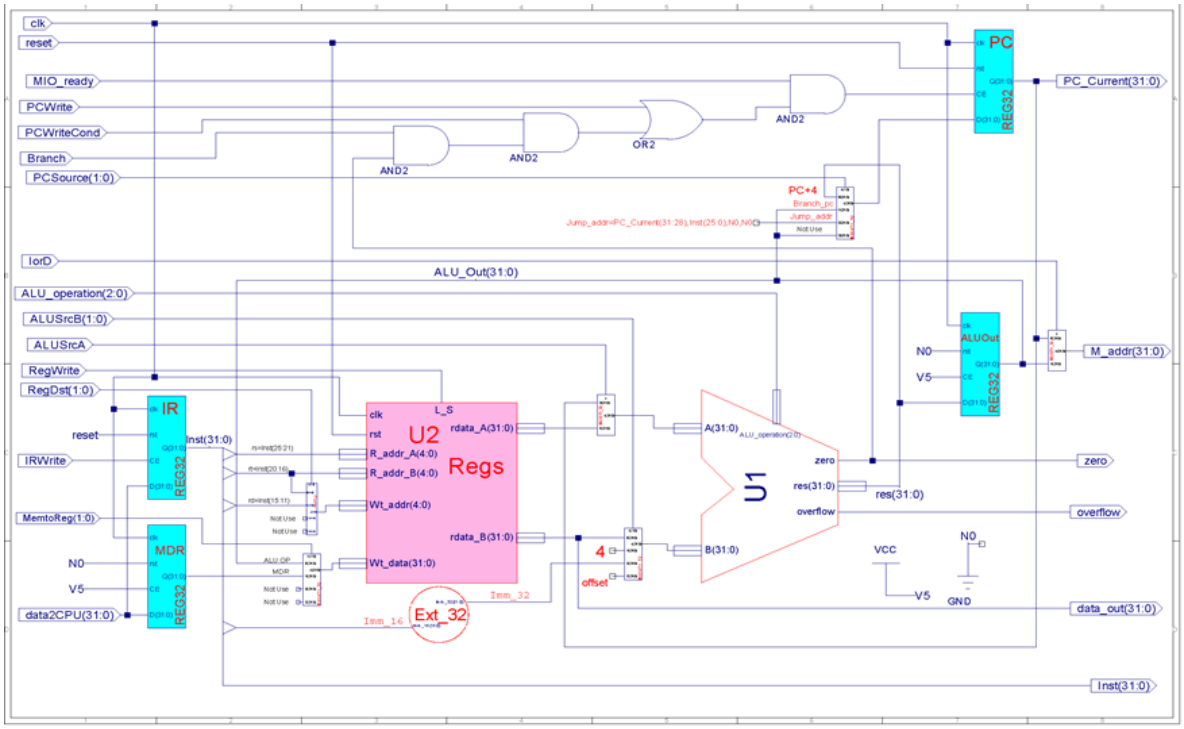
三、 实验过程和数据记录及结果分析

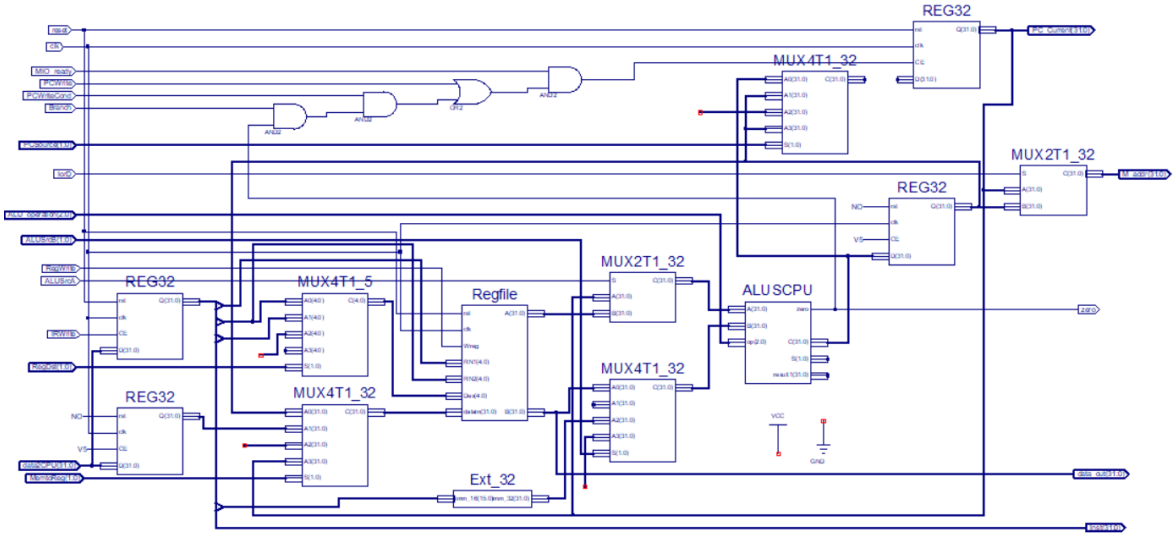
Construct the circuit

construct by myself:

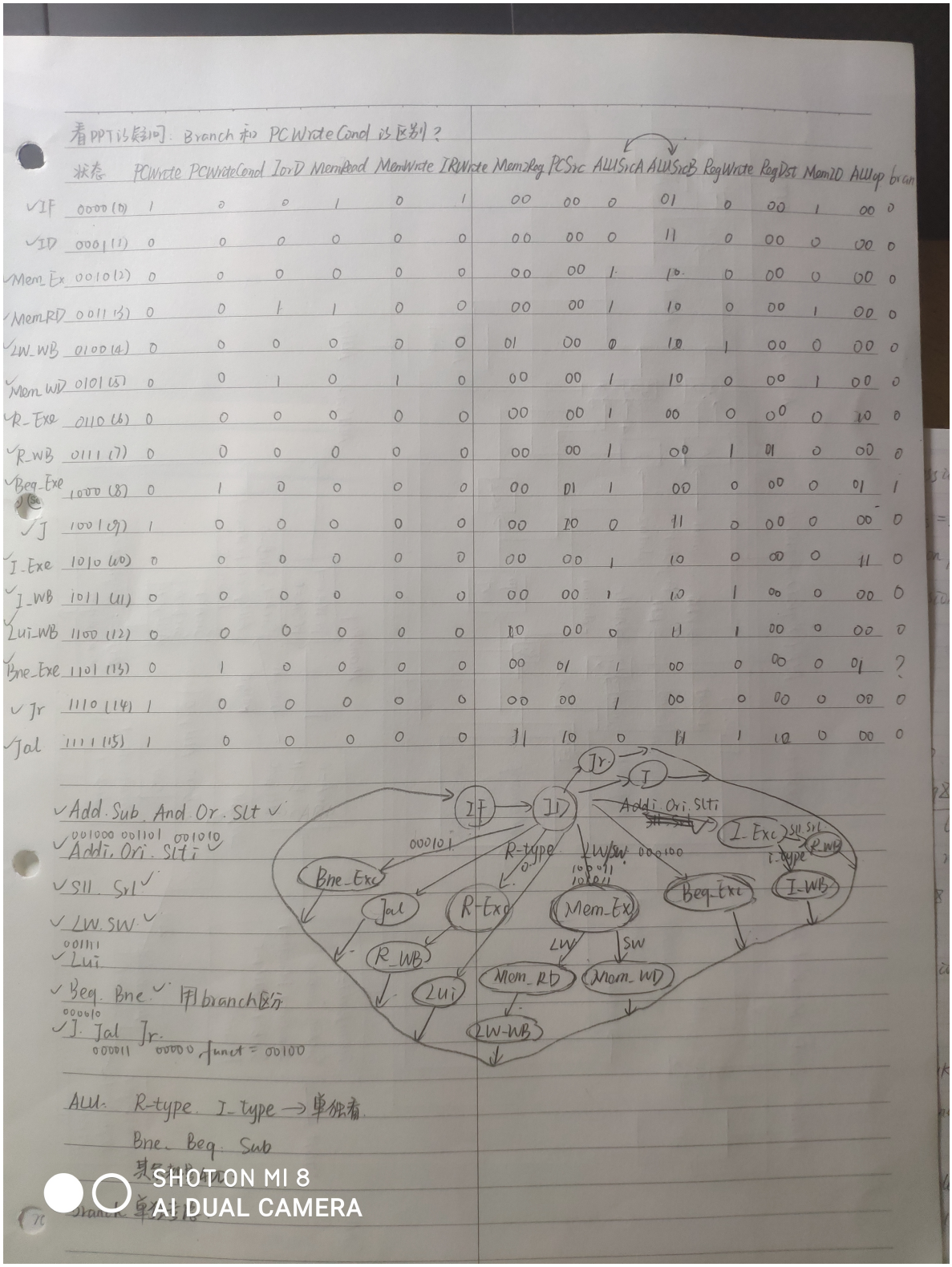


follow the lead of the slides:





Construct the Ctrl part



Modules

ALU_SCPU

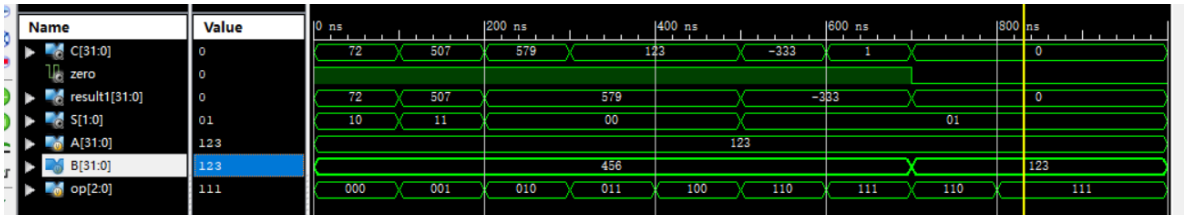
use the module ALU4b last semester

```
module ALU32b(
        input [31:0]A,
        input [31:0]B,
        input [1:0]S,
        output [31:0]C,
        output Co
    );
```

```
wire C0;
ALU4b m1(.A(A[3:0]),.B(B[3:0]),.S(S),.C(C[3:0]),.Ci(S[0]),.Co(C0));
wire C1;
ALU4b m2(.A(A[7:4]),.B(B[7:4]),.S(S),.C(C[7:4]),.Ci(C0),.Co(C1));
wire C2;
ALU4b m3(.A(A[11:8]),.B(B[11:8]),.S(S),.C(C[11:8]),.Ci(C1),.Co(C2));
wire C3;
ALU4b m4(.A(A[15:12]),.B(B[15:12]),.S(S),.C(C[15:12]),.Ci(C2),.Co(C3));
wire C4;
ALU4b m5(.A(A[19:16]),.B(B[19:16]),.S(S),.C(C[19:16]),.Ci(C3),.Co(C4));
wire C5;
ALU4b m6(.A(A[23:20]),.B(B[23:20]),.S(S),.C(C[23:20]),.Ci(C4),.Co(C5));
wire C6;
ALU4b m7(.A(A[27:24]),.B(B[27:24]),.S(S),.C(C[27:24]),.Ci(C5),.Co(C6));
ALU4b m8(.A(A[31:28]),.B(B[31:28]),.S(S),.C(C[31:28]),.Ci(C6),.Co(Co));

endmodule
```

```
module ALUSCPU(
input [31:0]A,
input [31:0]B,
input [2:0]op,  //operation
output [31:0]C,
output zero,
output [1:0]S,
output [31:0]result1
    );

wire [1:0]S=0;
assign S[0]=(op==6||op==1||op==7)?1:0;
assign S[1]=(op==0||op==1)?1:0;
wire co;
wire [31:0]result1; //ALU result
ALU32b m1(A,B,S,result1,co);

wire [31:0]result2;  //SLL result
wire [31:0]result3; //SRL result
wire [31:0]result4; //SLT result
assign result4=(result1[31]==1)?1:0;
SLL m2(A,B[10:6],result2);
SRL m3(A,B[10:6],result3);

wire [31:0]temp1;
wire [31:0]temp2;
assign temp1=(op==7)?result4:result1;
assign temp2=(op==4)?result3:result2;

assign C=(op==3||op==4)?temp2:temp1;
assign
zero=C[0]|C[1]|C[2]|C[3]|C[4]|C[5]|C[6]|C[7]|C[8]|C[9]|C[10]|C[11]|C[12]|C[13]|C
[14]|C[15]|C[16]|C[17]|C[18]|C[19]|C[20]|C[21]|C[22]|C[23]|C[24]|C[25]|C[26]|C[2
7]|C[28]|C[29]|C[30]|C[31];
endmodule
```

simulation:



Regfile

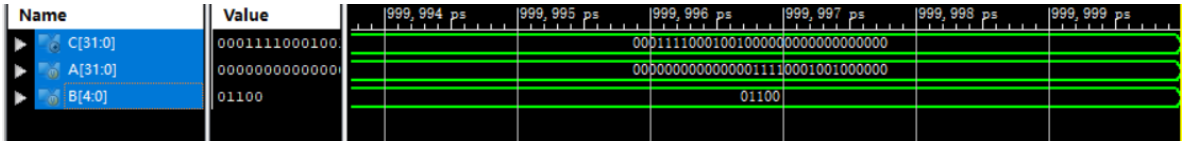
```
module Regfile(
input rst,
input clk,
input Wreg,
input [4:0]RN1,
input [4:0]RN2,
input [4:0]Des,
input [31:0]datain,
output [31:0]A,
output [31:0]B
    );
reg [31:0]a[1:31];
integer i;
assign A=(RN1==0)?0:a[RN1];
assign B=(RN2==0)?0:a[RN2];

    always @(posedge clk or posedge rst) begin
    if (rst==1) begin
        for (i=1; i<32; i=i+1)
        a[i] <= 0; // reset
    end
    else if ((Des != 0) && (Wreg == 1))
        a[Des] <= datain; // write
    end

endmodule
```

REG32

```
module REG32(
    input clk,
    input rst,
    input CE,
    input [31:0] D,
    output reg [31:0] Q
);

always @(posedge clk or posedge rst) begin
    if (rst) Q <= 32'h0000;
    else if (CE) Q <= D;
end

endmodule
```

SLL

In last experiment, I left SLL and SRL for a better solution. In this experiment, I try to calculate the 32 result of SLL from 0 to 31 and use a 32T1 MUX to output the asked one. I thought this method is not good and wish to learn a better way to do SLL from others.



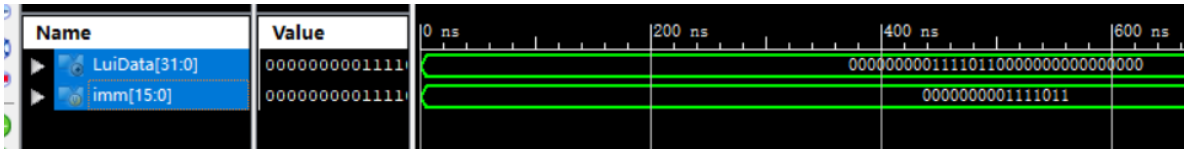
Ctrl

```
module ctrl(input  clk,
                input  reset,
                input  [31:0] Inst_in,
                input  zero,
                input  overflow,
                input  MIO_ready,
                output reg MemRead,
                output reg MemWrite,
                output reg[2:0]ALU_operation,
                output [4:0]state_out,

                output reg CPU_MIO,
                output reg IorD,
                output reg IRWrite,
                output reg [1:0]RegDst,
                output reg RegWrite,
                output reg [1:0]MemtoReg,
                output reg ALUSrcA,
                output reg [1:0]ALUSrcB,
                output reg [1:0]PCSource,
                output reg PCWrite,
                output reg PCWriteCond,
                output reg Branch
                );

reg [4:0] Q;
assign state_out = Q;

parameter IF = 5'b00000,
             ID = 5'b00001,
             EX_R = 5'b00010,
             EX_Mem = 5'b00011,
             EX_I = 5'b00100,
             Lui_WB = 5'b00101,
             EX_beq = 5'b00110,
             EX_bne = 5'b00111,
             EX_jr = 5'b01000,
             EX_JAL = 5'b01001,
             Exe_J = 5'b01010,
             MEM_RD = 5'b01011,
             MEM_WD = 5'b01100,
             WB_R = 5'b01101,
             WB_I = 5'b01110,
             WB_LW = 5'b01111,
             Error = 5'b11111;
```

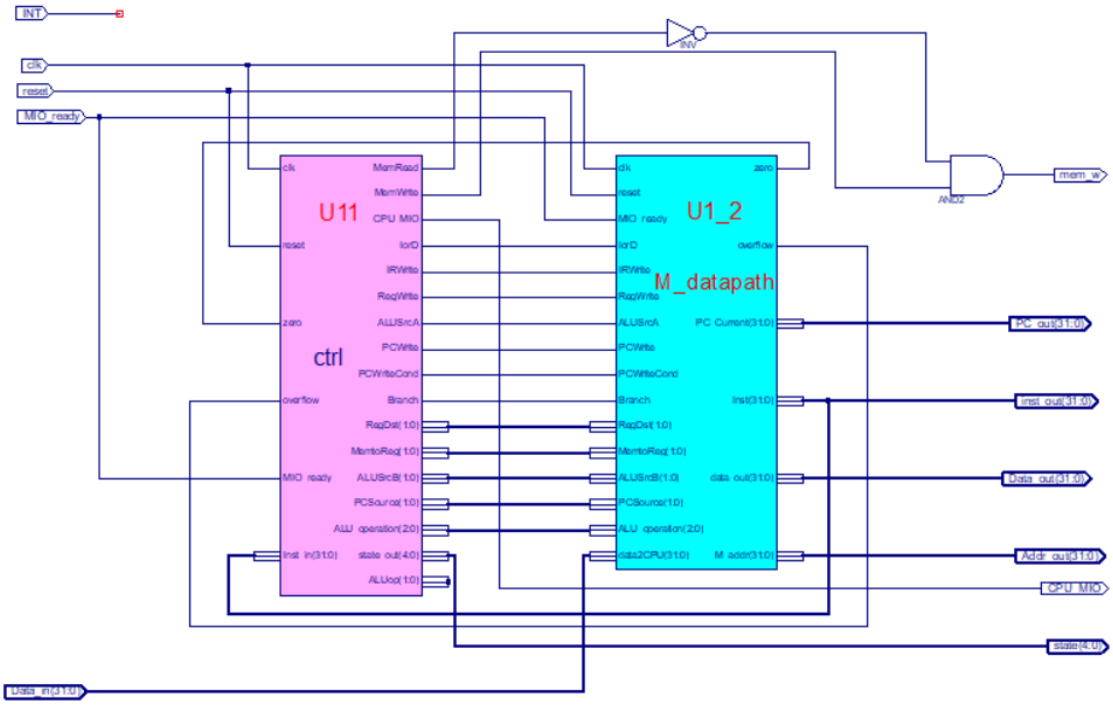
```
`define Datapath_signals{PCWrite, PCWriteCond,IorD, MemRead, MemWrite,IRWrite,
MemtoReg, PCSource, ALUSrcB,ALUSrcA,  RegWrite, RegDst, CPU_MIO}

parameter value0 = 17'h12821,   value1 = 17'h00060,
            value2 = 17'h00050, value3 = 17'h06001,
           value4 = 17'h00208,  value5 = 17'h05001,
           value6 = 17'h00010,  value7 = 17'h0001A,
           value8 = 17'h08090,  value9 = 17'h10160,
           valueA = 17'h00050, valueB = 17'h00058,
           valueC = 17'h00468, valueD = 17'h08090,
           valueE = 17'h10010, valueF = 17'h1076C;

parameter AND=3'b000, OR=3'b001, ADD=3'b010,
            SUB=3'b110, NOR=3'b100, SLT=3'b111,
                XOR=3'b011, SRL=3'b101;

always @ * begin
    case(Q)             //state
        IF: begin `Datapath_signals = value0; ALU_operation = ADD; Branch =
0;end
        ID: begin `Datapath_signals = value1; ALU_operation = ADD; Branch =
0;end
        EX_Mem: begin `Datapath_signals = value2; ALU_operation = ADD; Branch =
0;end
        EX_R:   begin `Datapath_signals = value6; Branch = 0;
                    case (Inst_in[5:0])
                        6'b100000: ALU_operation = ADD;
                        6'b100010: ALU_operation = SUB;
                        6'b100100: ALU_operation = AND;
                        6'b100101: ALU_operation = OR;
                        6'b100111: ALU_operation = NOR;
                        6'b101010: ALU_operation = SLT;
                        6'b000010: ALU_operation = SRL;       //shfit 1bit right
                        6'b000000: ALU_operation = XOR;
                        6'b001000: ALU_operation = ADD;
                        default:   ALU_operation = ADD;
                    endcase
                end
        EX_I:begin `Datapath_signals = valueA;  Branch = 0;
                case(Inst_in[31:26])
                    6'b001000: ALU_operation = ADD;     //Addi
                    6'b001100: ALU_operation = AND;     //Andi
                    6'b001101: ALU_operation = OR;      //Ori
                    6'b001110: ALU_operation = XOR;     //Xori
                    6'b001010: ALU_operation = SLT;     //Slti
                    default:   ALU_operation = ADD;
                endcase
             end//

        Lui_WB:begin `Datapath_signals = valueC; ALU_operation = ADD; Branch =
0;end//

        EX_beq:begin `Datapath_signals = value8; ALU_operation = SUB; Branch =
1; end//
        EX_bne:begin `Datapath_signals = valueD; ALU_operation = SUB; Branch =
0;end//
```

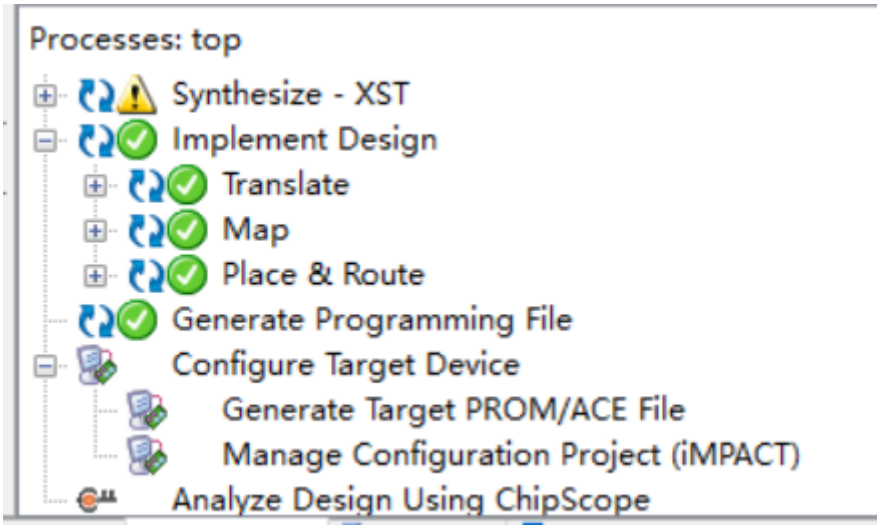
```
        EX_jr:begin `Datapath_signals = valueE; ALU_operation = ADD; Branch =
0;end//
        EX_JAL:begin `Datapath_signals = valueF; ALU_operation = ADD; Branch =
0;end//
        Exe_J:begin `Datapath_signals = value9; ALU_operation = ADD; Branch =
0;end//
        MEM_RD: begin `Datapath_signals = value3; ALU_operation = ADD; Branch =
0;end//
        MEM_WD:begin `Datapath_signals = value5; ALU_operation = ADD; Branch =
0;end//
        WB_R:begin `Datapath_signals = value7; ALU_operation = ADD; Branch =
0;end//
        WB_I:begin `Datapath_signals = valueB; ALU_operation = ADD; Branch =
0;end//
        WB_LW: begin `Datapath_signals = value4; ALU_operation = ADD; Branch =
0;end//
        default: begin `Datapath_signals = value0; ALU_operation = ADD; Branch =
0;end
    endcase
end

always @ (posedge clk or posedge reset) begin
        if (reset==1) Q <= IF;
        else
            case(Q)
                IF: begin if(MIO_ready) Q <= ID;
                    else Q <= IF;
                    end
                ID: begin
                        case(Inst_in[31:26])
                            6'b000000: begin
                                        case(Inst_in[5:0])
                                            6'b000000: Q<=EX_I; //SLL
                                            6'b000010: Q<=EX_I; //SRL
                                            6'b001000: Q <= EX_jr; // jr
                                            default: Q <= EX_R; // R
                                        endcase
                                    end
                            6'b000010: Q <= Exe_J; // j
                            6'b000011: Q <= EX_JAL;// jal
                            6'b000100: Q <= EX_beq; // beq
                            6'b000101: Q <= EX_bne; // bne
                            6'b001000: Q <= EX_I; // addi
                            6'b001010: Q <= EX_I; // slti
                            6'b001100: Q <= EX_I;// andi
                            6'b001101: Q <= EX_I;// ori
                            6'b001110: Q <= EX_I;// xori
                            6'b001111: Q <= Lui_WB;// lui
                            6'b100011: Q <= EX_Mem; // lw
                            6'b101011: Q <= EX_Mem; // sw
                            default: Q <= Error;
                        endcase
                    end
                EX_Mem: begin
                            case(Inst_in[31:26])
                                6'b100011: Q <= MEM_RD; // lw
                                6'b101011: Q <= MEM_WD; // sw
                                default: Q <= IF;
```

```
                                endcase
                        end
                EX_R:   begin
                            Q <= WB_R;
                        end
                EX_I:
                        if(Inst_in[31:26]==0)
                            Q<=WB_R;
                        else Q<=WB_I;
                Lui_WB: Q <= IF;
                EX_beq: Q <= IF;
                EX_bne: Q <= IF;
                EX_jr:  Q <= IF;
                EX_JAL: Q <= IF;
                Exe_J:  Q <= IF;
                MEM_RD: begin
                            //if(MIO_ready) Q <= WB_LW;
                            //else Q <= MEM_RD;
                            Q <= WB_LW;
                        end
                MEM_WD: begin
                            //if (MIO_ready) Q <= IF;
                            //else Q <= MEM_WD;
                            Q <= IF;
                        end
                WB_R:   begin
                            Q <= IF;
                        end
                WB_I:   begin
                            Q <= IF;
                        end
                WB_LW:  begin
                            Q <= IF;
                        end
                Error: Q <= Error;
                default: Q <= Error;
            endcase
end

endmodule
```



Top

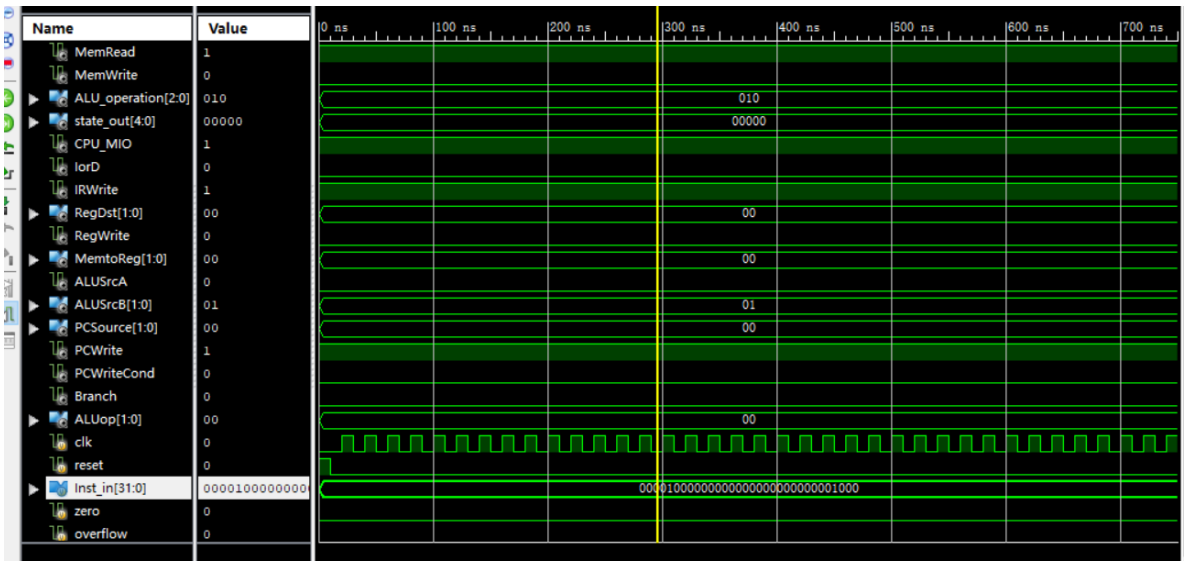




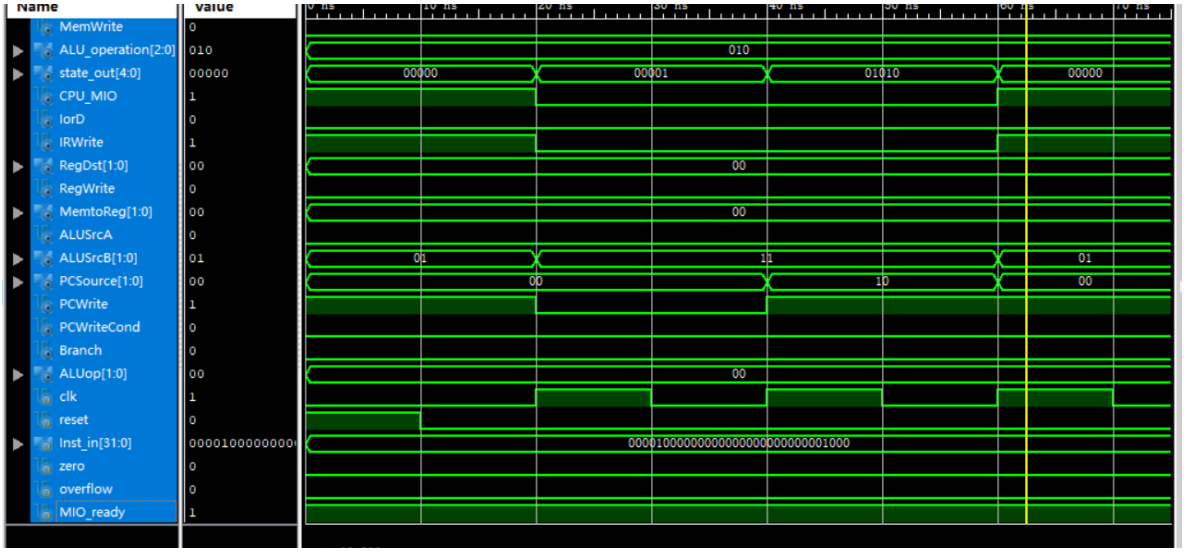

```
module test_contrl;

    // Inputs
    reg clk;
    reg reset;
    reg [31:0] Inst_in;
    reg zero;
    reg overflow;
    reg MIO_ready;

    // Outputs
    wire MemRead;
    wire MemWrite;
    wire [2:0] ALU_operation;
    wire [4:0] state_out;
    wire CPU_MIO;
    wire IorD;
    wire IRWrite;
    wire [1:0] RegDst;
    wire RegWrite;
    wire [1:0] MemtoReg;
```

```
    wire ALUSrcA;
    wire [1:0] ALUSrcB;
    wire [1:0] PCSource;
    wire PCWrite;
    wire PCWriteCond;
    wire Branch;
    wire [1:0] ALUop;

    // Instantiate the Unit Under Test (UUT)
    ctrl uut (
        .clk(clk),
        .reset(reset),
        .Inst_in(Inst_in),
        .zero(zero),
        .overflow(overflow),
        .MIO_ready(MIO_ready),
        .MemRead(MemRead),
        .MemWrite(MemWrite),
        .ALU_operation(ALU_operation),
        .state_out(state_out),
        .CPU_MIO(CPU_MIO),
        .IorD(IorD),
        .IRWrite(IRWrite),
        .RegDst(RegDst),
        .RegWrite(RegWrite),
        .MemtoReg(MemtoReg),
        .ALUSrcA(ALUSrcA),
        .ALUSrcB(ALUSrcB),
        .PCSource(PCSource),
        .PCWrite(PCWrite),
        .PCWriteCond(PCWriteCond),
        .Branch(Branch),
        .ALUop(ALUop)
    );

    initial begin
        // Initialize Inputs
        clk <= 0;
        reset = 1;
        Inst_in = 32'h08000008;
        zero = 0;
        overflow = 0;
        MIO_ready = 1;
        #10;
        reset=0;
        forever #10 clk<=~clk;

        #100;
        reset=1;
    end

endmodule
```

1.MIO_ready is always zero so CPU is not allowed to read instruction. Under this condition we can have a look at defaul status output.



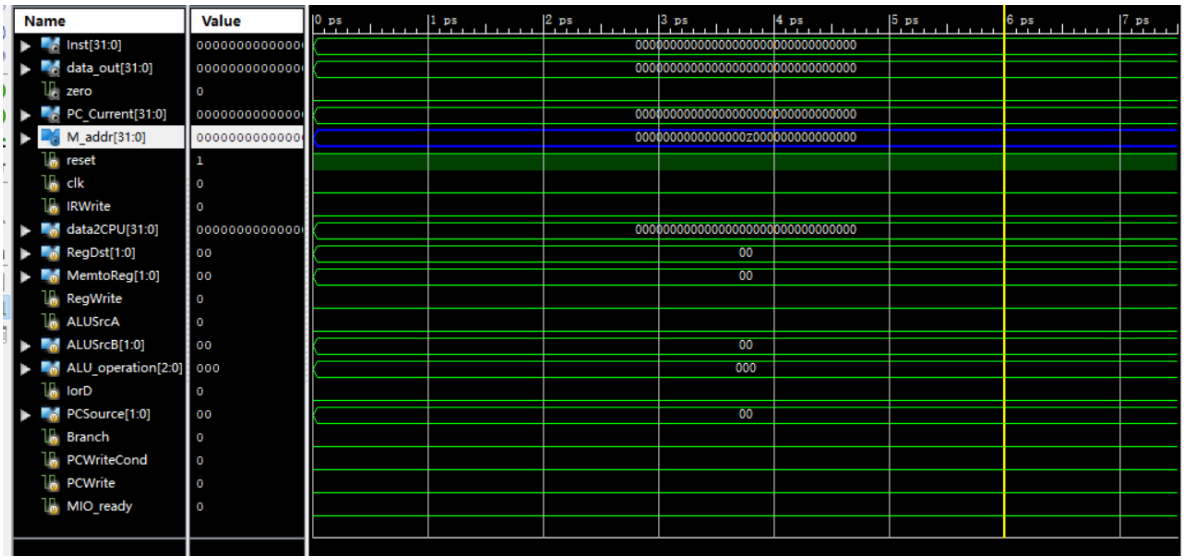
2.MIO_ready=1 for add instuction. The method to check the other instrument is just the same.




```
`timescale 1ns / 1ps

module M_datapath_M_datapath_sch_tb();

// Inputs
   reg reset;
   reg clk;
   reg IRWrite;
   reg [31:0] data2CPU;
   reg [1:0] RegDst;
   reg [1:0] MemtoReg;
   reg RegWrite;
   reg ALUSrcA;
   reg [1:0] ALUSrcB;
   reg [2:0] ALU_operation;
   reg IorD;
   reg [1:0] PCSource;
   reg Branch;
   reg PCWriteCond;
   reg PCWrite;
   reg MIO_ready;

// Output
   wire [31:0] Inst;
```

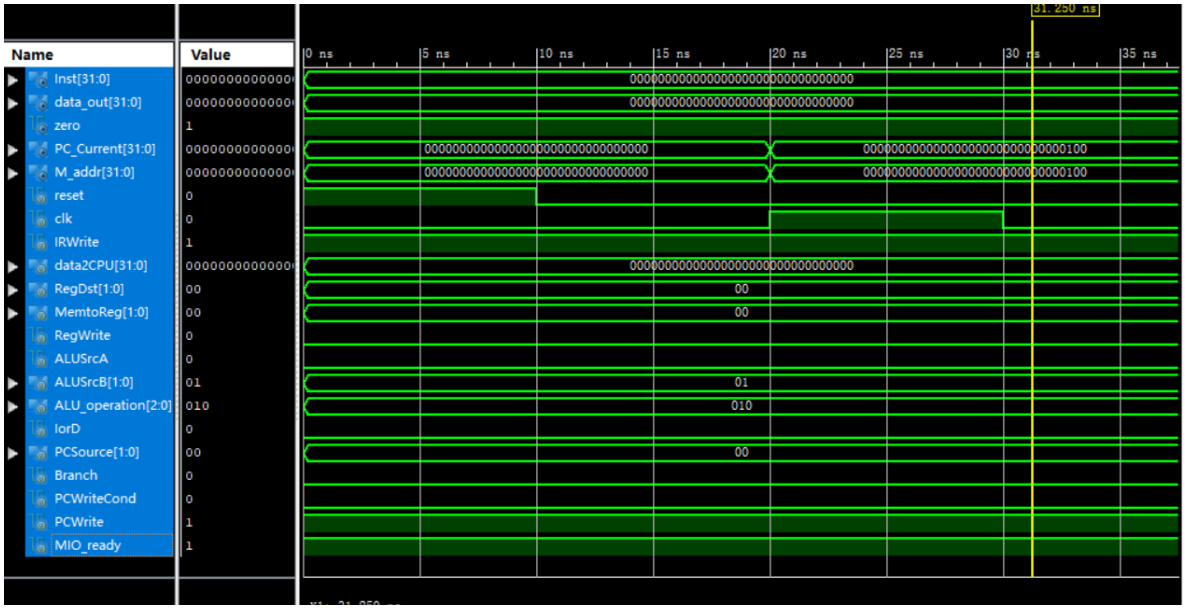
```
    wire [31:0] data_out;
    wire zero;
    wire [31:0] PC_Current;
    wire [31:0] M_addr;

// Bidirs

// Instantiate the UUT
    M_datapath UUT (
        .reset(reset),
        .clk(clk),
        .IRWrite(IRWrite),
        .data2CPU(data2CPU),
        .Inst(Inst),
        .RegDst(RegDst),
        .MemtoReg(MemtoReg),
        .RegWrite(RegWrite),
        .ALUSrcA(ALUSrcA),
        .ALUSrcB(ALUSrcB),
        .data_out(data_out),
        .ALU_operation(ALU_operation),
        .zero(zero),
        .PC_Current(PC_Current),
        .IorD(IorD),
        .M_addr(M_addr),
        .PCSource(PCSource),
        .Branch(Branch),
        .PCWriteCond(PCWriteCond),
        .PCWrite(PCWrite),
        .MIO_ready(MIO_ready)
    );
// Initialize Inputs
        initial begin
        reset = 1;
        clk = 0;
        IRWrite = 0;
        data2CPU = 0;
        RegDst = 0;
        MemtoReg = 0;
        RegWrite = 0;
        ALUSrcA = 0;
        ALUSrcB = 0;
        ALU_operation = 0;
        IorD = 0;
        PCSource = 0;
        Branch = 0;
        PCWriteCond = 0;
        PCWrite = 0;
        MIO_ready = 0;
        #10;
        reset=0;
        forever #10 clk=~clk;
        #100;
        reset=1;
    end
endmodule
```

1.test default status output to make sure the datapath correct. M_addr is blue because MIO_ready=0



2.test IF and ID instruction; the other instuctions are the same

```
initial begin
        reset = 1;
        clk = 0;
        IRWrite = 1;
        data2CPU = 0;
        RegDst = 0;
        MemtoReg = 0;
        RegWrite = 0;
        ALUSrcA = 0;
        ALUSrcB = 2'b01;
        ALU_operation = 3'b010;
        IorD = 0;
        PCSource = 0;
        Branch = 0;
        PCWriteCond = 0;
        PCWrite = 1;
        MIO_ready = 1;
        #10;
        reset=0;
        forever #10 clk=~clk;
        #100;
        reset=1;
end
```



四、 讨论与心得

In this experiment, I could design my own multi cycle CPU. The most difficult part is to figure out the transfer between states and fulfill the state transfer table. It's really difficult to find out errors in the table and design a check method that is efficient. When it comes to wiring, actually, after doing the wiring job in the previous exeperiment, this part is not so difficult now. I have found some easier way to design the schematic and became much more familliar with ise software.

I also met some problems in the experiment. One of them that I haven't understand now is the wire design between Branch and PCWriteCond, I can't really understand whether it is meaningful to depart these two signal. In my mind, they actually have the same function. Then there's another question about how to simulate, I can't think of a solution good enough now. Maybe I have to find a more guaranteed methond to do simulation before final checking.

As the deadline is postphoned, I'v found some mistakes in the lab and correct them in this report. And I also solved the problem of SLL/SRL, I asked teachers and classmates for how they construct SLL and SRL and get these two better answer:

```
module srl32(
    input [31:0] A,
    input [31:0] B,
    output reg [31:0] res
);
    always @* begin
        res = A;
        if (B >= 5'b10000) res = 0;
        else begin
            if (B[4]) res = {16'b0, res[31:16]};
            if (B[3]) res = {8'b0, res[31:8]};
            if (B[2]) res = {4'b0, res[31:4]};
            if (B[1]) res = {2'b0, res[31:2]};
            if (B[0]) res = {1'b0, res[31:1]};
        end
    end
endmodule
```

```
module SLL(
    input [31:0] A,
    input [31:0] B,
    output [31:0] res
);

assign res=A<<(B[4:0]);
endmodule
```

I never know that verilog can support sentences like "assign res=A+B" or "assign res=A&B", that makes me regret for all the time I sppend to debug my own ALU...